UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/026,266	12/21/2001	Michael L. Fraenkel	RSW920010208US1	7861

46320 7590 05/20/2005

CHRISTOPHER & WEISBERG, PA
200 E. LAS OLAS BLVD
SUITE 2040
FT LAUDERDALE, FL 33301

EXAMINER	
STEELMAN, MARY J	

ART UNIT	PAPER NUMBER
2191	

DATE MAILED: 05/20/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/026,266	FRAENKEL ET AL.	
	Examiner	Art Unit	
	Mary J. Steelman	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *29 December 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-20* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-20* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *21 December 2001* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. This Office Action is in response to amendment and remarks received 29 December 2004. Per Applicant's request the Specification has been amended. Claims 1-3 and 17 have been amended. Claims 1-20 are pending. Applicant is reminded that claim notation, Patent Rules 1.121, 5 (c), should be 'Original', not 'Originally Filed'.

Drawings

2. In view of the amendment to the Specification, the prior objection to the drawings is hereby withdrawn.

Specification

3. In view of the amendments to the specification, the prior objections are hereby withdrawn.

Claim Rejections - 35 USC § 101

4. In view of the amendments to claims 1-3, the prior 35 USC 101 rejection is hereby withdrawn.

Claim Rejections - 35 USC § 112

5. In view of the amendments to the claims, the prior 35 USC 112 second paragraph is hereby withdrawn.

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person

having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. Claims 1-20 are rejected under 35 U.S.C. 103(a) as being unpatentable over US Patent Application Publication 2004/0015936 A1 to Susaria et al., in view of US Patent 6,601,114 B1 to Bracha et al.

Per claim 1:

A custom class loader apparatus configured to dynamically locate and load classes in a virtual machine in accordance with an associated dependency specification, the custom class loader comprising:

-class loading logic configured to specifically and dynamically locate, define and load a class specified by name;

(Susaria : [0056], "Each module in the application may be associated with its own class loader...")

-a list of peer class loaders arranged in accordance with the associated dependency specification and, list generation logic configured to generate said list when said specified class has been replaced or when said dependency specification has been modified;

(Susaria : [0056], "The class loader module may include a hierarchical stack of class loaders. Each class loader may have one parent class loader and zero or more child class loaders...")

-a flag indicating whether said class has been replaced;

(Susaria : [0059], ...application may include a dirty class monitor...")

Regarding:

-deference logic configured to defer said location, definition and loading of said specified class to said peer class loaders in said list.

Susaria disclosed:

[0057], "...class loaders that are each configured to load one or more classes for the application when invoked." Class loaders in the hierarchy (peers) contain the logic to load specific classes.)

Susaria disclosed (Abstract) a dynamic class loader. The class loader module may include a hierarchical stack of class loaders. Susaria failed to specify a list of peer class loaders. However, a hierarchical stack of class loaders is a data structure linking related elements, with a similar effect to 'listing peer class loaders'. Peer class loaders can be identified, as they have a common parent node in the hierarchical data structure. Susaria failed to specify "list generation logic", generated when said specific class has been replaced or when said dependency specification has been modified. Although Susaria failed to generate a "list", he did disclose that when a class was replaced / modified, that dependent classes are provided with appropriate class loaders. The hierarchical tree relates to dependencies. Susaria failed to explicitly specify "a flag comprises a dirty bit" to indicate a class has been replaced. However, he did provide a dirty monitor that noted a changed class [0059], which provides the same information. It is inherent that a 'dirty class monitor' provides some type of indication (flag) when a 'dirty' condition exists. When a "Dirty Class Monitor" notifies that a replacement loader is needed, the "Application Class Loader Controller" finds a peer loader to load the modified class and suitable loaders for any dependent classes. [0059], "The application may detect that a class has been changed...may include a dirty class monitor that may monitor classes used by the application

and detect when any of the classes have been changed. The class loader for the class may be replaced with a new version of the class loader..."

Susaria disclosed dynamic class loading, but failed to explicitly specify "deference logic configured to defer". However, Bracha disclosed (FIG. 4 & Col. 11, lines 40-46), "In lazy loading...a class is not loaded until it is needed during execution (deference logic)." Col. 15, lines 9-11, "The constraints written or otherwise recorded for use by the VM...are enforced, e.g. checked, when and if referenced class B is actually loaded.", col. 15, lines 29-31, "...a VM can implement fully lazy loading with verification."

Therefore, it would have been obvious, to one of ordinary skill in the art, to consider the effects of Susaria's 'dirty class monitor' to inherently suggest a 'dirty flag' and for the hierarchy of class loaders to provide the details for a 'list of peer loaders' and 'dependencies' among loaders. It would have been obvious, to one of ordinary skill in the art, at the time of the invention to modify Susaria's invention, by including Bracha's suggestion that a dynamic loader may be 'lazy loaded'. One would be motivated to modify Susaria's dynamic loader because (Bracha, col. 15, lines 31-42) the advantages include: The behavior of a program with respect to linkage errors is the same on all platforms and implementations. One need not anticipate all the places where a class or method might be linked and attempt to catch exceptions at all those places. Users can determine the availability of classes or other modules in a reliable and simple way...a programmer can test for the availability of classes on a platform in a reliable and simple way."

Per claim 2:

-said flag comprises a dirty bit.

(Susaria : FIG. 8 & [0142-0143], "Dirty Class Monitor",[0059], "The application may detect that a class has been changed...a dirty class monitor...")

Per claim 3:

-custom class loader conforms to the specification of a Java programming language version 1.2 delegation-style custom class loader.

(Susaria : [0081], "Follow a delegation mechanism that is a modification of the mechanism described in JDK, version 1.2.")

Per claim 4:

-receiving a request to load a specified class;

(Susaria : FIG. 7, [0100], "...it may forward the "load class" request to the class loader controller", [0106], "load class requests")

-determining whether said specified class has been replaced;

(Susaria : [0088], "If a class changes, all the classes that use this class may be reloaded as well...", [0100], "The class loader controller then may determine which class loader is supposed to load the class.")

-if it is determined that said specified class has been replaced, constructing a new instance of the class loader and generating a list of peer class loaders to which location, definition and loading

of said specified class are to be deferred in accordance with a dependency specification in the virtual machine;

(Susaria : [0100], "Any notification for a class change may also come to the class loader controller so that it can recreate the concerned class loaders.", [0108], "Receiving a notification from the replacement logic of the application class loader controller when a class changes.", [0110], ...application class loader controller may also be responsible for dispatching the "load class" requests to the appropriate class loader.", [0141], "The class loader controller may then replace the class loader with the new class loader. If there are one or more classes that depend on the class to be reloaded, the class loaders responsible for reloading the dependent classes may be located and replaced as well..." Also see FIG. 7, #308 regarding dependencies.)

-deferring said location, definition and loading to said peer class loaders in said list.

(0141], "the class loader controller may invoke each of the necessary class loaders to reload the classes that need to be reloaded in response to the change in the class.")

Per claim 5:

-determining step comprises checking a dirty bit in the class loader.

(Susaria : [0140], "dirty class monitor that may monitor classes used by the application and detect when any of the classes have been changed.")

Per claim 6:

-traversing each peer class loader in said dependency specification;

(Susaria : FIG. 7, [0138], "application may include a class loader module that may include a hierarchical stack of class loaders that are each configured to load one or more classes for the application when invoked.", [0141], "The class loader controller may then locate the class loader responsible for loading the class in the hierarchical stack of class loaders.")

-adding a reference for each said traversed peer class loader to said list.

Per claim 7:

-dependency specification comprises a tree of nodes, each said node encapsulating a reference to a dependency of said specified class, one of said nodes encapsulating a reference to said specified class.

(Susaria : [0141], "The class loader controller may then replace the class loader with the new class loader. If there are one or more classes that depend (dependency) on the class to be reloaded, the class loaders responsible for reloading the dependent classes may be located and replaced as well.")

Per claim 8:

-beginning with said one node encapsulating a reference to said specified class, traversing each node in said dependency specification using a depth-first traversal strategy until encountering either a leaf node or a node encapsulating a reference to a dependency already referenced in said list;

-responsive to said encountering, traversing each node in said dependency specification using a breadth-first traversal strategy until encountering said node encapsulating said reference to said specified class;

-adding a reference for each traversed node to said list.

(See response to claim 7 above. Class loaders are located in the hierarchical stack of class loaders. In some cases one loader may load multiple classes. In some cases there may be a specific loader associated with a specific class. Classes that are dependent upon a changed class, may also need to be reloaded with a modified class loader.

Per claim 9:

-adding at least one reference to a peer class loader to said list based upon a corresponding reference stored in a list of peer class loaders identified in one of said traversed peer class loaders.

(Susaria : [0145], "Helper (utility) classes in one module may be symbolically referenced by other module classes...helper classes may be loaded by the same class loader (peer class loader)...)

Per claim 10:

-setting said dirty bit responsive to said specified class being replaced.

(Susaria : [0142-0143], "Dirty Class Monitor FIG. 8 illustrates a dirty class monitor...Tasks related to the dirty class monitor may include, but are not limited to, registration and

notification." It is inherent that a "dirty class monitor" has a technique to indicate via a 'bit' that a class has been modified.)

Per claim 11:

-setting each dirty bit in each peer class loader referenced in said list responsive to said specified class being replaced.

(Susaria : [0142-0143] & FIG. 8)

Per claim 12:

(See limitation addressed in claim 4 above.)

Per claim 13:

(See limitation addressed in claim 5 above.)

Per claim 14:

(See limitations addressed in claim 6 above.)

Per claim 15:

(See limitations addressed in claim 7 above.)

Per claim 16:

(See limitations addressed in claim 8 above.)

Per claim 17:

-traversing at least one parent class loader associated with the class loader through to a primordial class loader;

(Susaria : [0088], "if the system class loader is assumed to be at the highest level (primordial class loader), then a change in a system class may trigger class reloading in all the lower levels...")

-adding a reference for each said traversed parent class loader to said list.

(Susaria : [0056], "The class loader module may include a hierarchical stack of class loaders. Each class loader may have one parent class loader and zero or more child class loaders. Each module in the application may be associated with its own class loader...")

Per claim 18:

-adding at least one reference to a parent class loader to said list based upon a corresponding reference stored in a list of parent class loaders identified in one of said traversed parent class loaders.

(Susaria : [0056], "The class loader module may include a hierarchical stack of class loaders. Each class loader may have one parent class loader and zero or more child class loaders. Each module in the application may be associated with its own class loader...")

Per claim 19:

(See limitations addressed in claim 10 above.)

Per claim 20:

(See limitations addressed in claim 1 above.)

Response to Arguments

8. Applicant has argued, in substance, the following:

(A) As Applicant has pointed out on page 14, 3rd paragraph, of Remarks, received 29 December 2004, "Susaria does not support a prima facie case of obviousness..."

Examiner's Response: In response to applicant's argument that Susaria does not support a prima facie case of obviousness, the test for obviousness is not whether the features of a secondary reference may be bodily incorporated into the structure of the primary reference; nor is it that the claimed invention must be expressly suggested in any one or all of the references. Rather, the test is what the combined teachings of the references would have suggested to those of ordinary skill in the art. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981). One of ordinary skill in the art would be motivated to consider a hierarchical list of loaders to provide the same advantage as a 'list of class loaders', as both are data structures, linking children (peers) to a parent loader. One of ordinary skill in the art would be motivated to consider a 'dirty monitor' to inherently provide an indication (a flag) that a class has been modified. One of ordinary skill in the art would be motivated to consider that an invention for 'dynamic loading' could include the concept of 'lazy loading' as provided by Bracha, because it minimizes effort by loading only

what is needed. Deferred loading was known in the art at the time of the invention. Thus the Susaria / Bracha combination do support a prima facie case of obviousness.

(B) As Applicant has pointed out on page 16, last paragraph, "Susaria...fails to...suggest a list of peer class loaders arranged in accordance with the associated dependency specification, and list generation logic configured to generate said list when said specified class has been replaced or when said dependency specification has been modified."

Examiner's Response: See rejection of claim 1 above. Peer class loaders are arranged in hierarchical order in the Susaria invention. The hierarchical ranking is in accordance with the associated dependency specification. The 'dirty monitor' has logic to alter the hierarchical order when a class has been modified. Note [0105] – [0108], "dynamic class reloading mechanism...may create the application class loader controller and use it to load the classes. The container may interact with the application class loader controller for...registering as a class change listener (when dirty monitor indicates a changed class)...Receiving a notification from the replacement logic of the application class loader controller when a class changes..."

(C) As Applicant has pointed out on page 17, first paragraph, "the dirty class monitor...cannot be equated to a flag when giving the term flag its ordinary and plain meaning."

Examiner's Response: Inherently the 'dirty class monitor' has a mechanism (a flag) that indicates the class has changed.

(D) As Applicant has pointed out on page 17, first and last paragraph, "...no where in Susaria is it ever suggested that any logic 'defers' any operation to a peer class loader in a list..."

Examiner's Response: Susaria disclosed 'dynamic class loading', but failed to explicitly disclose deference logic. Examiner has supplied a secondary reference, Bracha, that provides the suggestion of 'lazy loading', loading classes as needed.

Conclusion

9. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Mary Steelman, whose telephone number is (571) 272-3704. The examiner can normally be reached Monday through Thursday, from 7:00 AM to 5:30 PM If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached at (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Mary Steelman 05/05/2005